CLAIMS:

1   1. A method for analyzing software code comprising the steps of:
2   a) automatically generating program graphs representing said code utilizing
3   static analysis techniques;
4   b) automatically applying a set of rules to said program flow analysis graphs;
5   c) automatically identifying potential software problems from rules set analysis
6   results; and,
7   d) reporting said software problems where one or more of best practices
8   violations and coding errors may occur. control and data flow analysis

1   2. The method according to Claim 1, wherein said rules set represents one or more
2   selected from the group comprising: use of best practices and common coding errors,
3   or combinations thereof.

1   3. The method according to Claim 1, wherein said reporting d) includes presenting the
2   results in the context of corresponding source code or object code.

1   4. The method according to Claim 1, wherein step b) includes performing rule
2   searches applied to said program graphs.

1   5. The method according to Claim 1, wherein said software code subject to said static
2   analysis techniques comprises one or more selected from the group comprising: object
3   code, source code, a compiler intermediate representation, of said software code, and
4   other program representations, or combinations thereof.

1   6. The method according to Claim 3, wherein a program graph includes a control
2   analysis graph, said static analysis technique automatically generating said control
3   analysis graphs from said software code.

1 7. The method according to Claim 3, wherein a program graph includes a data flow

2 analysis graph, said static analysis technique automatically generating said data flow

3 analysis graph from said software code.

1 8. The method according to Claim 3, wherein a program graph includes an

2 intraprocedural control graph, said static analysis technique automatically generating

3 said intraprocedural control graphs from said software code.

1 9. The method according to Claim 3, wherein a program graph includes an

2 interprocedural control graphs, said static analysis technique includes automatically

3 generating said interprocedural control graphs from said software code.

1 10. The method according to Claim 5 wherein said static code analysis further

2 includes automatically identifying classes, fields, methods and class attributes, said set

3 of rules being further applied to said classes and class attributes.

1 11. The method according to Claim 5 wherein said static code analysis further

2 includes automatically identifying attributes of classes, methods, fields, and aspects of

3 a program's body.

1 12. The method according to Claim 5, wherein said step b) further includes the step

2 of: receiving said program graphs and class attributes information and performing a

3 graph rewriting technique.

1 13. The method according to Claim 12, wherein a result of applying graph rewriting

2 includes generating a run-time characteristics model for said program.

1 14. The method according to Claim 12, wherein said step b) further includes the step

2 of receiving said program graphs and attributes information, and performing a

3 reachability analysis.

YOR920020352US1   15

1   15. The method according to Claim 14, wherein said reachability analysis is

2   performed with or without constraints.

1   16. The method according to Claim 14, further comprising the step of employing a

2   rule search engine to automatically apply a set of rules to said rewrite graph results,

3   reachability analysis results and attributes to identify one or more selected from the

4   group of: possible performance errors or problems concerning correctness, security,

5   privacy and maintainability of said software code.

1   17. The method according to Claim 14, wherein said rewrite graph technique includes

2   traversing a program graph to locate nodes containing attributes of interest and to

3   locate edges to add or remove from said program graph.

1   18. The method according to Claim 17, wherein said reachability analysis includes

2   traversing the program graphs and adding or removing edges to extend or reduce

3   reachability, respectively.

1   19. The method according to Claim 18, wherein a rule is applied to determine whether

2   a node representing a particular method is reachable by traversing said graph from a

3   particular head node, said head node being user selectable.

1   20. A static analysis framework for analyzing software code, said framework

2   comprising:

3   means for automatically generating program graphs;

4   rule search engine for automatically applying a set of rules to said program

5   graphs;

6   means for automatically identifying potential software problems from rules set

7   analysis results; and,

8   means for reporting said problems to enable correction of instances where one

9   or more of best practices violations and common coding errors may occur.

1   21. The static analysis framework as claimed in Claim 20, wherein said rules set

2   represents one or more selected from the group comprising: use of best practices and

3   common coding errors, or combinations thereof.

1   22. The static analysis framework as claimed in Claim 20, wherein said software code

2   comprises scalable componentized applications according to a software development

3   platform.

1   23. The static analysis framework as claimed in Claim 18, wherein said program

2   graphs include one or more selected from the group comprising: a control analysis

3   graph, a data flow analysis graph, an intraprocedural control flow graph and an

4   interprocedural control flow graph, said static analysis technique automatically

5   generating a respective one of said control analysis graph, data flow analysis graph,

6   intraprocedural control flow graph and interprocedural control flow graph from said

7   software code.

1   24. The static analysis framework as claimed in Claim 23, further including means for

2   automatically identifying classes, fields, methods and class attributes, said set of rules

3   being further applied to said classes and class attributes.

1   25. The static analysis framework as claimed in Claim 23, wherein said static code

2   analysis further includes automatically identifying attributes of classes, methods,

3   fields, and aspects of a program's body.

1   26. The static analysis framework as claimed in Claim 20, wherein said means for

2   automatically generating program graphs includes means for performing graph

3   rewriting.

1 27. The static analysis framework as claimed in Claim 26, wherein results of said
2 graph rewriting include a run-time characteristics model for said program.

1 28. The static analysis framework as claimed in Claim 26, wherein said means for
2 automatically generating program graphs includes: means for performing a
3 reachability analysis, said reachability analysis being performed with or without
4 constraints.

1 29. The static analysis framework as claimed in Claim 28, wherein said rule search
2 engine automatically applies a set of rules to said rewrite graph results, reachability
3 analysis results and attributes to identify one or more of: possible performance errors
4 or problems concerning correctness, security and privacy of said software code.

1 30. A computer program device readable by a machine, tangibly embodying a
2 program of instructions executable by a machine to perform method steps for
3 analyzing software code, said method steps comprising:
4   a) automatically generating program graphs representing said code utilizing
5 static analysis techniques;
6   b) automatically applying a set of rules to said program graphs;
7   c) automatically identifying potential software problems from rules set analysis
8 results; and,
9   d) reporting said software problems to enable correction of instances where
10 one or more of best practices violations and common coding errors may occur.